Inferring Software Update Practices on Smart Home IoT Devices Through User Agent Analysis

Vijay Prakash
New York University
New York, USA
vijay.prakash@nyu.edu

Sicheng Xie
New York University
New York, USA
sx810@nyu.edu

Danny Yuxing Huang
New York University
New York, USA
dhuang@nyu.edu

ABSTRACT

Smart home IoT devices are known to be breeding grounds for security and privacy vulnerabilities. Although some IoT vendors deploy updates, the update process is mostly opaque to researchers. It is unclear what software components are on devices, whether and when these components are updated, and how vulnerabilities change alongside the updates. This opaqueness makes it difficult to understand the security of software supply chains of IoT devices.

To understand the software update practices on IoT devices, we leverage IoT Inspector's dataset of network traffic from real-world IoT devices. We analyze the User Agent strings from plain-text HTTP connections. We focus on four software components included in User Agents: cURL, Wget, OkHttp, and python-requests. By keeping track of what kinds of devices have which of these components at what versions, we find that many IoT devices potentially used outdated and vulnerable versions of these components—based on the User Agents—even though less vulnerable, more updated versions were available; and that the rollout of updates tends to be slow for some IoT devices.

CCS CONCEPTS

• **Security and privacy** → **Mobile and wireless security**; • **General and reference** → Empirical studies.

KEYWORDS

IoT; supply chain; updates

ACM Reference Format:
Vijay Prakash, Sicheng Xie, and Danny Yuxing Huang. 2022. Inferring Software Update Practices on Smart Home IoT Devices Through User Agent Analysis. In *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED '22), November 11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3560835.3564551

1 INTRODUCTION

Smart home technologies, also known as smart devices or Internet-of-Things (IoT) devices, are gaining popularity, such as smart TVs, speakers, cameras, and medical devices, yet they are breeding grounds for security and privacy threats [8, 29], using vulnerable software components [3] and putting the users and other hosts on the network at risk.

To mitigate these risks, like software, many IoT devices offer updates that can be installed automatically or manually by users. However, this update process remains largely opaque; it is unclear what software components are on devices, whether/how any software updates have occurred on these software components, and how vulnerabilities change alongside the updates across a large number, variety, and the long tail of IoT devices in smart homes [1]. This opaqueness makes it hard to keep track of the software bill-of-materials (SBOMs) across devices over time and understand the software supply chain of smart home IoT devices in general.

One of the reasons for this knowledge gap is the lack of scale in many lab-based analyses of IoT devices. To understand the software components on IoT devices and the update behaviors, researchers often analyze the firmware binaries. However, commercial smart home IoT devices tend to use proprietary and/or protected software components (such as the firmware and libraries) that are difficult to extract and reverse-engineer [33]. Furthermore, researchers often need the physical IoT devices for their analyses. The number of devices that can be studied is often constrained by time and budget; one of the largest sample of IoT devices studied in the lab is about 120 devices [39, 42] and does not cover many devices in the long tail [1]. As such, it is difficult for researchers to identify whether and how IoT devices update to patch vulnerabilities over time.

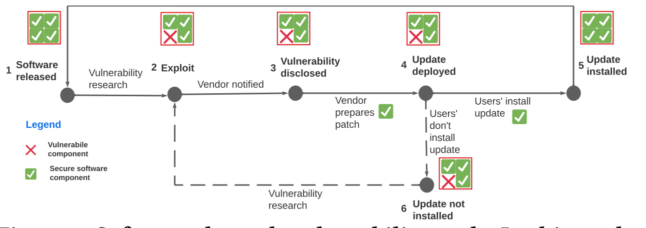

Figure 1: Software through vulnerability cycle. In this study, we are focusing on the updates, step 4, 5 and 6.

Figure 1 shows software through vulnerability life cycle [47]: 1) A software is released; 2) Security researchers find vulnerabilities; 3) Researchers ethically notify vendors about the vulnerabilities privately and give a grace period of 90 days (current standard) to fix those issues and release updates; 4) Vendor publicly discloses the vulnerability along with updates containing fixes; 5) In the ideal case, users download the updates and update the software making it secure against the exploits; 6) In the worst case, users don't install the update and software stays vulnerable. Finally the cycle continues starting from vulnerability research again after both the 5th and 6th stage of the cycle.

Objectives. Our goal is to understand software update practices on IoT devices in real-world smart homes: what software components and versions are present on devices at a particular time, as well as when the versions and the associated vulnerabilities have changed over time (i.e., likely updates). This knowledge will shed light on the update practices across a large number and variety of IoT devices and vendors, thus paving the way for deeper empirical understanding of IoT software supply chain security.

Method. To achieve this goal, we conduct the first known longitudinal analysis of IoT software components in real-world smart homes and provide an initial understanding of update practices in the IoT ecosystem. In particular, we leverage IoT Inspector's dataset of network traffic crowdsourced from smart homes around the world [23]. The dataset includes the likely identities of devices, along with any User-Agent strings extracted from plain-text HTTP headers sent by these devices. The User Agents indicate *potential* use of certain software components and their versions. We would like to note that devices from our dataset do not use plain HTTP for the majority of communication, but they do send them out for some reason with empty payload, possibly to test a connection or discovery purposes (based on our lab studies). We have not decrypted the HTTPS connections while collecting data for this study.

In this paper, we analyze the versions of four different software components that appear in the User Agent strings: cURL (which is both a library, `libcurl`, and a binary), Wget (a binary), OkHttp library, and Requests (Python's) library. We observe these components in the User Agent strings on 23,837 devices (including 359 what looks like IoT devices) across 4,562 real-world smart homes between Apr 2019 and Oct 2021.

Findings. We show that update practices in the IoT ecosystem are different from (sometimes worse than) findings from general purpose computing devices. In particular, none of the IoT devices in our dataset included the latest versions of the four software components in the User Agent, even when the latest versions were available at the time IoT Inspector captured the data. In some cases, the lower versions included critical CVEs that could have been fixed if the IoT devices had been using the latest versions at the time. Furthermore, we find that vendor deploy updates in rolling fashion, and oftentimes rolled out updates are not the latest. In other words, when vendors deploy updates, they do not always update the software component to the latest versions, which means oftentimes devices are left in vulnerable state even after end user install the update.

These findings paint a grim picture of the slow (and sometimes the lack of) software update practices on certain IoT devices. Such update practices could be a result of users not promptly installing updates, IoT vendors not deploying updates in time, or both—it is an open question what is the cause for our observation. It is our hope that this study will offer the first evidence on the update practices on IoT devices, highlight the associated software supply chain risks, and provide the impetus for more secure practices on users, IoT vendors, and regulators.

2 RELATED WORK

Software updates. In the last couple of decades, there has been a push from academia and industry to improve the software update hygiene for all kinds of software because old software has been correlated with compromise [12, 27], so it is in the best interest of everyone's security and from economical standpoint to use the latest version of software [9, 14, 16, 34, 46]. Propelled by this push, major software vendors have adopted the practice of deploying frequent updates, which improved the vendor update deployment practice. On the other hand, end users are advised to follow "Best Practices", which includes updating software as soon as updates are available if users can [12, 27, 46]. For software like operating systems (OS) and applications running on them, previous studies have shown that user update installation practice can be modeled with either geometric or exponential distribution, which means majority of the users update the software after new release, but rest of them take longer duration [26, 40, 44, 46].

Updating software is considered to be a part of "Best Practices" to increase security by shrinking the vulnerable state of hosts, [16, 41] recommend updates. Khan et al. [27] found a positive correlation between infection indicators and a lack of regular updating practice, [12] showed correlation in devices not updating and being compromised at some point of time. Sarabi et al.[46] found that frequent discovery of vulnerabilities in a software limits the benefit of providing faster updates. DeKoven et al. [16] measure the correlation between "Best Practices" and impact on security risk.

Four factors for update. One way to define security of a host is how vulnerable it is; the less vulnerability a host has, the more secure it likely is. From the software update perspective, there are four factors that decide the vulnerable state of a host: 1) how quickly users install the updates; 2) how quickly vendors deploy the patch; 3) how vendors deploy the patch; and 4) how frequently vulnerabilities are found [46]. These combined together tell about update practice. So far studies have looked at all the four factors for software that are used for computing in general (i.e. mobiles, computers, tablets, etc.) [6, 7, 9, 10, 13, 14, 18, 35, 36, 40, 44–47, 49].

There have been studies about all the factors affecting the vulnerability state of a host. These studies have been conducted for both clients side (general purpose computing devices) and server side hosts. For general purpose computing devices, there is [16, 46] on user update behaviors using various software; [6, 7, 9, 10, 14, 35, 36, 45, 47] on vendor update deployment and vulnerability disclosure; [17, 20, 21] on mechanism of deploying patches where they suggest silent updates are the most effective. On servers side hosts update behaviors there are [18, 26, 40, 44, 49], where they measure patch installation delays after major security incidents and routine patch releases after vulnerability disclosures.

Update practice for IoT. Specific to IoT devices, Yousefnezhad et al.[51] has done a comprehensive survey of IoT product lifecycle, including vulnerability management and software updates. There have been numerous study about IoT update infrastructure, i.e., how to deliver updates to IoT devices logistically [28, 30, 50, 52], and securely [11]. It has also been found that user awareness about updating devices has improved — IoT device users prefer to buy products that guarantees updates [34].

So far we are not aware of any study about the current update practice in IoT ecosystem. The case for IoT devices is different from major software vendors because of the sheer amount of devices and vendors available in the market [1]. Another possible reason for the differentiation could be that vendor nudges to update software could not reach IoT device users, unlike computing devices if users are actively using devices they see the update nudge on UI. Because some IoT devices can only be interacted with companion apps, and users might forget to use the apps as these devices might be always on without user interaction. Also, unlike few major software vendors for personal computing, IoT vendors are fragmented and too many [1]. All this makes it difficult to apply the results of previous studies to the IoT ecosystem.

3 METHOD

3.1 Inferring software components from User Agents

To understand software update practices on real-world IoT devices, we use IoT Inspector's [23] dataset of network traffic, crowdsourced from global smart homes. The dataset includes names and manufacturers of devices, along with various network header data, including the User Agent strings extracted from plain-text HTTP connections. Note that IoT Inspector does not capture the contents of the HTTP traffic; nor does it collect personally identifiable information [23]. We look at a subset of this data, which consists of 4,562 different smart homes containing 23,837 unique devices (including IoT and general purpose computing) from Apr 9, 2019 to Oct 12, 2021.

The captured User Agent strings offer indications for software components that devices have *likely* used. We focus on four software components used as User Agents: cURL [15], Wget [48], Python Requests [43], and OkHttp [38]. We picked these because they are the most popular (by the number of devices) User Agents that are not browsers. This information is by no means the ground truth for what software components devices *actually* used (since HTTP clients can, in theory, put anything into the User Agent field). Still, we assume that devices' HTTP clients are honest about being non-browsers. There is evidence for non-truthful User Agents for browsers [2], but we are unaware of any evidence of non-truthful User Agents for non-browsers like cURL, Wget, Requests, and OkHttp. As such, throughout this paper, we use the User Agent as a proxy for the actual software components.

IoT Inspector captures the HTTP User-Agent header seen in the network traffic from uniquely identified devices with a timestamp of when it was seen. Devices whose traffic is captured could include general purpose computing devices (computers, phones, etc.) and IoT devices. We distinguish the device categories with the Fingerbank [19] APIs (which identify devices based on the MAC address and destination hosts contacted over a proprietary ML model).

To find out how old and vulnerable versions of these four software are used, we manually compile a list of versions released with dates from their websites or code bases. Each version is assigned a release number from 0 (first release) to N (last release). We also compile a table of vulnerabilities each version of these four software have. Vulnerabilities in each version are either available on their website or we compile it by searching names of each software on the National Vulnerability Database (NVD) [37] and use the CVE number and Common Platform Enumeration (CPE) to map them to different versions.

3.2 Metrics for understanding update practices

To get the understanding of update practices, from the collected data where HTTP user agent is one of the four software, and compilation of four software version release dates and vulnerability each version have, we curate statistical metrics for all the unique device and user agents combination. This includes following:

- What software is used as user agent and its used version
- First and last seen timestamp in the IoT Inspector dataset
- At the last seen time, the latest available version of that software
- At the last seen time, the number of versions and days (or months) behind this used user agent version is from this latest available version at this time
- At the last seen time, the number of vulnerabilities that could have been avoided by using the latest version. We explain how this is calculated later in the next subsection 3.3.

For our analysis to gain perspective of update practices we rely on three metrics retroactively:

(1) number of avoidable vulnerabilities, having vulnerabilities directly contribute to the vulnerable state of a host adversely
(2) number of versions behind
(3) and number of days behind (directly correlated with number of versions behind), they both show how quickly hosts are updated which involves both user promptness in installing updates and vendors deploying updates.

From our dataset we can't distinguish if users are not installing updates quickly or vendors are deploying updates slowly, but we can tell about the update practices comprising both with these three metrics.

3.3 How these metrics are calculated

Versions behind for a user agent when it's seen in our data is calculated by subtracting the release number of the used version from release number of most recent available version at this seen time. To calculate the number of avoidable vulnerabilities, we take the difference between the set of all the vulnerabilities in the used version and the set of vulnerabilities in the most recent version, which tells us vulnerabilities that were in the used version but not in the latest version, i.e., they were fixed in the latest version. Let's say, X be the set of vulnerabilities in the used version and Y be the set of vulnerabilities in the latest available version. Number of avoidable vulnerabilities would be (X - Y), whereas this - (subtraction) represents the difference of set operation.

For example, let's suppose cURL version 7.20.0, which is release # 112, released on 2010-02-09, and has 10 vulnerabilities as of today. Let's also suppose we see a device with the user agent cURL 7.20.0 on 2020-02-09 in the dataset. Latest available version of cURL on 2020-02-09 is version 7.68.0, which is release # 187, released on 2020-01-08, and has 8 vulnerabilities as of today. The number of versions behind for this seen version is 75 and the number of days behind is 3620 (9 years and 11 months). For the sake of this example, let's assume 8 of the vulnerabilities in 7.20.0 are not present in 7.68.0 and 2 of them were still present on 2020-02-09, which also means 7.68.0 has 6 new vulnerabilities introduced in it but at some point after its release maybe even after the date 2020-02-09. So, the number of vulnerabilities that could have been avoided by using 7.68.0 on 2020-02-09 is 8 because they are fixed in this version.

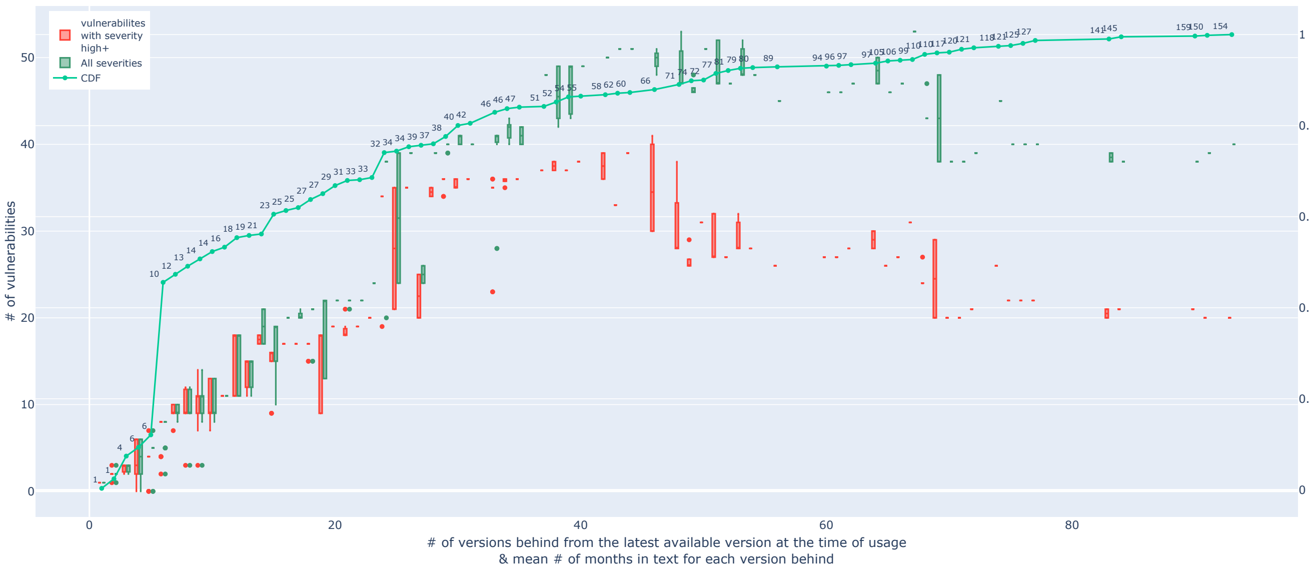

Figure 2: Number of avoidable vulnerabilities in cURL against number of versions behind, and CDF distribution of devices for number of versions behind.

4 FINDINGS

4.1 Outdated software components introduce avoidable vulnerabilities

Focusing on three factors of update practice, excluding the way vendors deploy patches, our finding for all kinds of client hosts (including general purpose computing and IoT devices) is consistent with previous studies. Nappa et al. [35] finds that devices move on to the latest version slowly showing a long tail, and Sarabi et al. [46] models how long they stay vulnerable (if they don't update) using geometric distribution which also shows long tail. Our findings using the three metrics in the graph Figure 2 and A.1 (Figure 5) for the four software over our capture duration suggest a similar trend, which tells us that our data is valid and devices are not lying about the user agents seen in our data.

In Figure 2, cURL is used as a user agent. On the horizontal axis we have a number of versions behind for different versions of user agents seen at different times during our data collection. The text label above the CDF for the version is the mean number of months older version is from the latest version available at the time of usage, for example, when a used version is 80 release versions behind, from the latest version it's on average 140 months (approximately 12 years) old. On the vertical axis on left, we have the distribution of the number of vulnerabilities that could have been avoided for every version behind, for e.g., when version behind is 25, number of avoidable vulnerabilities are in the range 21-35 (median 28) for critical and high severity vulnerabilities, and 24-39 (median 31.5) for vulnerabilities of all severities. On the vertical axis on right, we have a CDF plotted for the number of versions behind for 621 unique devices and cURL combinations. As per our data for cURL, there is a 50% chance that when cURL is used as a user agent it's at least 14 release versions behind, 14 months old, and it could have avoided a median of 11 vulnerabilities by using the latest available version of that time. For cURL, at the long end of the tail we see 3 devices using more than 90 release (~12 years) old versions.

Plot for all the versions shows a similar long tail of older release version usage for all four software; see Figures 2, and A.1 (Figure 5). The number of avoidable vulnerabilities is different and depends upon how many vulnerabilities a software component has. OkHtttp and Python Requests don't have many vulnerabilities, so updating to the latest version affects the vulnerability state to a lesser degree. From the plot of cURL and Wget, we can say that outdated versions over a period of time accrue more vulnerabilities and expand the host's vulnerability state. The previous statement is not true if a software doesn't have vulnerabilities, evident from the plot of Python Requests, and OkHttp.

We see surprising downtrend in number of avoidable vulnerabilities after number of version behind is more than 50 for cURL. cURL releases new version every 49 (median) days. So 50 and 90 version behind cURL is approximately 7 to 12 years old. Presumably the reason for the downtrend could be that cURL had less features 7 years ago, as its code base grew number of vulnerabilities increased.

4.2 Update practices of IoT devices are different from general computing devices

To find the distinction in update practice (including vendor deployment and user installation) of IoT devices versus non-IoT devices we categorize devices in six categories as per Fingerbank's [19] APIs. These six categories are: "IoT platforms", "IoT non-platforms" , "Computing", "Storage and Printers", "Networking", and "Others"; see Table 1.

We distinguish between "IoT platforms" and "IoT non-platforms". If an IoT device allows third-party applications to be installed on them, we call them "IoT Platform". We distinguish "IoT platforms" from "IoT non-platforms" because applications installed on them could be affecting the version of user agents captured from those devices. "Computing" category includes general purpose computing devices. "Storage and Printers" category contains network storage devices (NAS) and printers. "Networking" category contains networking devices found in homes. Rest of the devices that can't be

Table 1: Device types in our 6 device categories based on Fingerbank API.

Category	Device types
IoT platforms	smart TVs, set top boxes (STB), and digital video recorders (DVR)
IoT non-platforms	rest of the IoT devices, such as smart cameras, smart assistants, speakers, smart vacuum cleaners
Computing	computers, phones, tablets, and Raspberry Pi kind of devices
Storage and Printers	network storage devices (NAS), and printer
Networking	routers, wireless access points (WAP), switches, and firewalls
Other	rest of the devices that can't be categorized by Fingerbank

classified by Fingerbank API are in Other category. Example of few vendors seen in our data are in Table 2.

Table 2: Examples of few vendors we have seen for four software in our data.

Software component	Few example vendors
cURL	Samsung SmartThings, Google Nest, Parrot IoT, Vizio TV, WD TV Live, Meraki WAP, Synology NAS, Roomba, etc.
OkHttp	Amazon Alexa and Fire TV, Google Chromecast and Home, Roomba, Vizio TV, NVidia Shield, Belkin Router, etc.
Wget	WD TV Live, Netgear, Synology NAS, QNAP NAS, Amazon Alexa, Tablo DVR, etc.
Python Requests	Eero WAP, Cisco WAP, Synology NAS, TP-Link, NVidia Shield, Telldus Smart home, etc.

We box plot the distribution of vulnerabilities avoided (affects the vulnerability state of a device), number of versions behind, and number of days behind for each user agent seen in the earlier mentioned six categories for all the four software in Figures 3, and Appendix A.2. We would like to note that for OkHttp, Figure 4 shows that older versions (60 months old) and newer versions (20 - 25 months old) are equally vulnerable. This is because OkHttp has only two vulnerabilities and they exist in versions from 2017, and 2019. So either used versions of OkHTTP are five years old (after 2014), or two years old (after 2017) — they both contain two vulnerabilities and are equally vulnerable.

4.2.1 Comparing "Computing" group with IoT devices. From Figures 3, and Appendix A.2 (Figure 6), we could see that each category has a different distribution than others. By looking at the mean and median number of avoidable vulnerabilities, we can clearly see that personal computing devices have better software update posture than both the IoT device categories for all the four software, except for cURL in "IoT non-platforms" category. We looked further into this case, and found that there are 105 unique Samsung SmartThings [25] devices present in our dataset out of 163 devices in the "IoT non-platforms" category. These SmartThings devices consistently use the same cURL version 7.60.0, and are seen in a small burst of time. At the time when they are seen they are only 6 versions behind from the latest available version of cURL. These devices single-handedly improve the update posture of "IoT non-platforms" devices seen in our data by reducing the mean and median number of avoidable vulnerabilities in this category. Focusing on the update practice of SmartThings over our data collection duration, we find their update practice is not good — they have two kinds of devices and both of them use the same version of cURL for more than 20 months. We conclude that these devices might have released an update at the time when we started data collection, and their presence in large numbers skewed the results for "IoT non-platforms" category. If we exclude SmartThings devices, in the case of cURL, computing devices' update practice is better than "IoT non-platforms" and "IoT platforms" combined together. See mean and median values of number of avoidable vulnerabilities with and without SmartThings devices for these categories for cURL in Table 3.

Table 3: Number of avoidable vulnerabilities in "Computing", "IoT platforms", and "IoT non-platforms" groups of cURL.

Group	mean	median
Computing	15.54	16.0
IoT platforms	30.66	34.0
IoT non-platforms	10.22	8.0
IoT non-platforms without Samsung SmartThings	15.18	13.5
Both IoT groups together	16.83	8.0
Both groups together without Samsung SmartThings	24.76	28.0

4.2.2 Comparing "IoT platforms" group with "IoT non-platforms". Focusing on IoT devices only, both the categories perform similarly in terms of updates practice if we take a look at the mean and median values of avoidable vulnerabilities for two IoT categories for all four software in the Appendix B (Table 5), except for cURL for the reason explained in previous subsection. "IoT non-platforms" are doing better in general, which could be probably because large amount of third-party applications on platforms don't have as consistent update practice as a single vendor IoT devices.

Takeaway: As previous studies have only focused on general purpose computing devices, we find that update practice of IoT devices are worse than computing devices.

4.3 Possible slow vendor deployment and end user installation in IoT devices

In general purpose computing, prior work [17] has suggested that silent updates lead to quicker update installation compared to non-silent updates, and vendor update deployments depend upon vulnerability disclosure [35]. For the software we used in our study, vulnerability disclosure factor for update practices is the same for all categories of devices, but the IoT group is using older versions for longer duration. That means the remaining three factors — user promptness, vendor deployment, and mechanism of deployment — are contributing to the reason why update practice of IoT devices is worse than general purpose computing devices. As these software are unlikely to be directly used by end users on IoT devices,

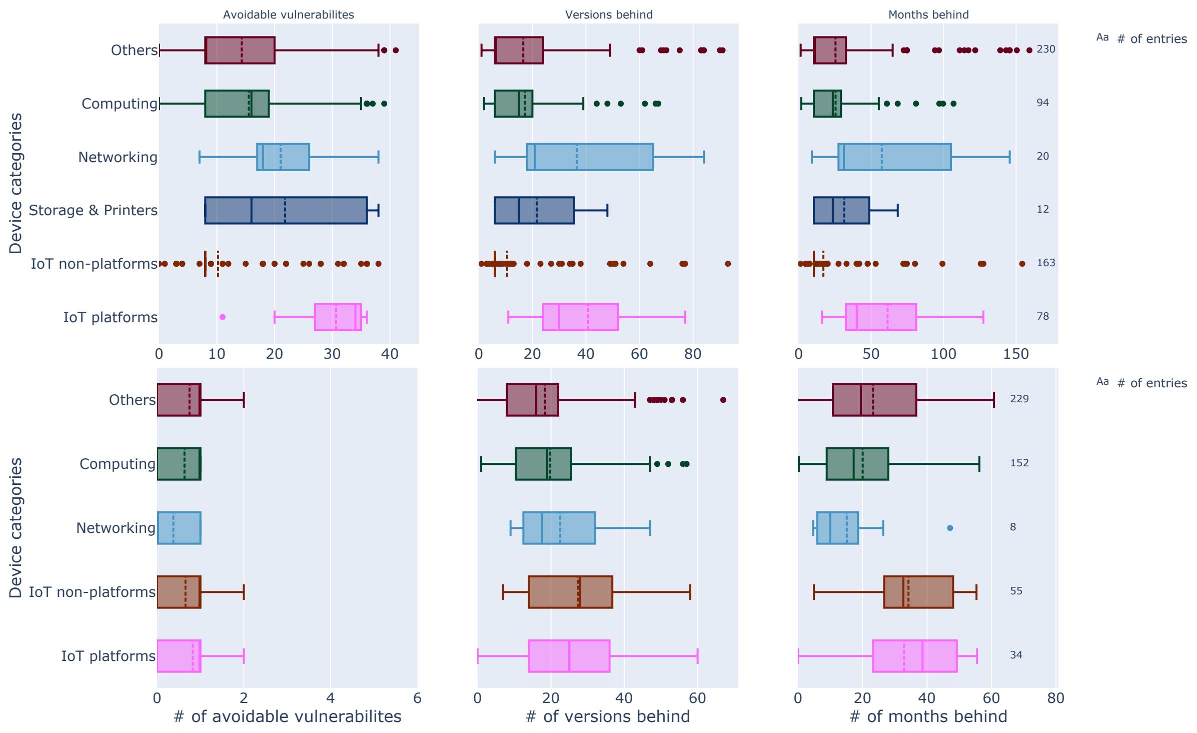

Figure 3: Distribution of number of avoidable vulnerabilities, number of versions behind, and number of months behind for all categories of IoT devices using cURL (top), and OkHttp (bottom).

updates for them typically go through the IoT vendors. Regarding vendor deployment factor in update, there has been an argument that vendors are bad for security, because they add an additional pit stop for the patches getting to end users [31, 32]. So maybe vendors are deploying patches slower than the computing group. User promptness to install patches is affected by update mechanism deployed by the vendors. The majority of well performing software component used on devices in computing groups have now adopted the silent update strategy [17]. Maybe IoT vendors are deploying patches promptly but not using silent update mechanisms, presumably for reasons like, not being able to do silent updates as IoT devices have constraints on computational power. In contrast with computing groups, some IoT devices are always on and don't need user engagement after the setup. To update IoT devices users have to use the companion apps, it could also be possible that users set up the IoT device and forget to check the companion app, which could be another reason for users' not promptly installing updates. A future work would be finding data relevant to all these factors to help us understand the actual reason behind sluggishness in IoT group update practices.

4.4 IoT vendors not deploying latest versions

To understand the vendor deployment factor of update practice we plotted versions of four software used by user agent of different devices from different vendors over the duration of our data capture. We only used devices from "IoT non-platforms" for these plots because "IoT platforms" allow third party apps. Versions seen from "IoT platforms" category devices will not depict a single vendor's deployment practice. For both the vertical subplots in the Figure 4, and A.3 (Figure 7) on the vertical axis we have different versions of software used, and time on the horizontal axis. All software is used by user agent of different vendors so we manually picked vendors which were seen more frequently in our data. Different colors of distribution represent different vendors as mentioned on the legends on the bottom right corner of every figure. The latest available version with respect to time is plotted in green color to give an understanding how far behind these used versions are from latest version. Plots also mention the metrics number of avoidable vulnerabilities and number of versions behind, described in section 3.3, in square brackets next to plotted distributions.

From these previous plots we can see the trend that user agents of different devices of the same vendors are using the same version over a period of time. We also see that some devices from a vendor have updated to a newer version while some of them are still using older versions. This could possibly be explained by either IoT vendors are deploying updates in a rolling fashion or users are not installing the updates after the vendor deploy the updates. We do see a jump in versions after a period, which is shown on the right side subplot. We explain the right side plot in the next subsection.

4.5 Time to repair (TTR)

We also try to calculate time to repair per version in terms of days for each software by calculating the number of days between two subsequent versions seen in our data and divide it by the number of release versions differences between them. A TTR is the slope between two points of the same vendor on the right side subplot.

Right subplots in Figure 4 only include points when time and version both were seen in increasing order only. To elaborate, let's

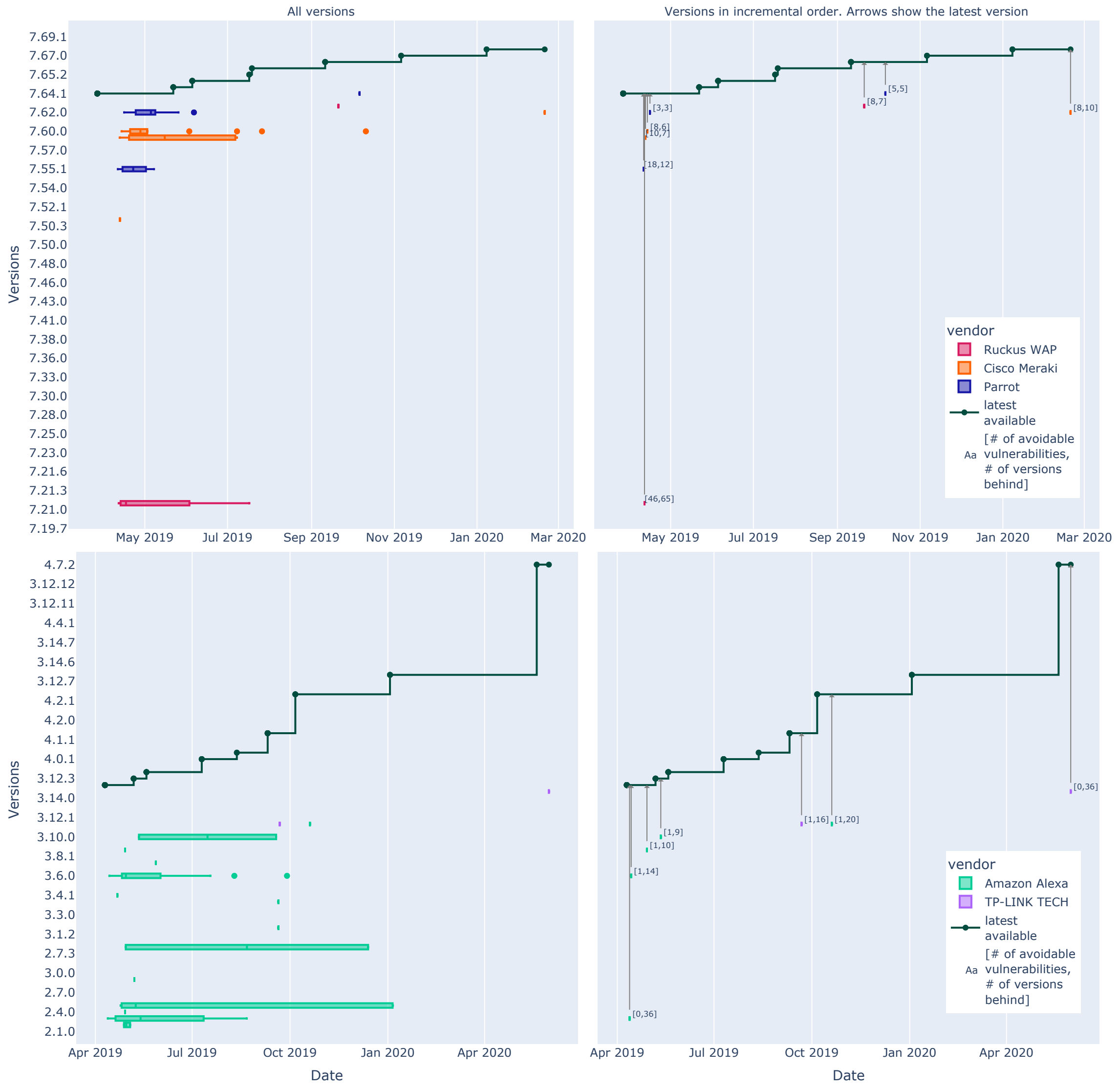

Figure 4: For different vendors using cURL (top), and Wget (bottom), distribution of versions used against time in left subplot; version seen in incremental order with respect to time, along with number of avoidable vulnerabilities and number of version behind in square brackets in right subplot. Both the subplots show latest available version at the time of version usage.

suppose that in April 2019, we see Alexa [24] devices using version 2.4.0 of OkHttp on dates 1st, 10th, and 15. Again in 2019, we see older version 2.3.0 in May, a newer version 3.5.0 in July on dates 1st and 15th, and another newer version 2.5 on July 20th being used by Alexa devices. We include 2.4.0 used on 1st April and 3.5.0 used on 1st July in the right subplot because this is the first time we have seen these versions. Clearly suggesting that the vendor has deployed a patch as per our data. We don't include 2.3.0, used in May because the device has not been updated to a newer version, as we have seen newer versions from Alexa devices before on 1st April. We don't include 2.5.0 used on July 20th even though it's an increment in version from 2.4.0 because it might have been rolled out by the vendor earlier, sometime between 1st April and 1st July, it's just not present in our data, as we see 3.5.0 being used before on July 1st.

Using this methodology we attempt to calculate the TTR per version of each vendor and report it in the Table 4. As our dataset is small, we have only few points in the right subplot in Figures 4. So TTRs we have calculated are few in terms of numbers. In the Table 4, the first small TTR for each version is because vendors probably rolled out the updates shortly after we started the capture, but the second ones are probably closer to actual TTR for the respective vendor. From this data we can say TTR for different vendors for the same software are different, consistent with [35].

Takeaways from our findings:

- IoT vendors are slower at deploying updates than all four software components, which could leave end users vulnerable if software has patched vulnerabilities in deployed updates.
- Even when IoT vendors do deploy updates they don't update to the latest version.

Table 4: Calculated TTR per version in days, hours:minutes:seconds for few vendors.

software	vendor	TTR per version
cURL	Cisco Meraki	1 days, 9:44:01
	Cisco Meraki	104 days, 0:30:41
	Parrot	0 days, 12:26:02
	Parrot	57 days, 20:46:07
	Ruckus WAP	2 days, 13:43:00
Python requests	eero WAP	0 days, 22:45:37
	eero WAP	320 days, 13:54:33
OkHttp	Alexa	0 days, 1:39:02
	Alexa	3 days, 16:50:06
	Alexa	6 days, 12:20:04
	Alexa	80 days, 9:25:09
	TP-Link Tech	63 days, 6:40:32

- In case of cURL, the updated version could have avoided vulnerabilities by updating to the latest version available.

5 DISCUSSIONS AND FUTURE WORK

Despite the small number of software components studied, our analysis paves the way for future studies. Hopefully our preliminary findings are enough to point out to the research community that more effort is needed to gain understanding of update practice in the IoT ecosystem. We are planning to collect more data in future so that we can get more consistent, statistically large, and informative results, including (i) more device identification information (as we currently rely on FingerBank's API, which uses a blackbox machine learning model with unclear accuracy); and (ii) more evidence of software components beyond user agents (e.g., fingerprinting network traffic with p0f to infer the operating system [4], or with TLS ClientHello to infer the SSL library and versions [3]).

Three factors out of four that are collectively responsible for update practice: user promptness, vendor update deployment, and mechanism of update deployment, we couldn't tell exactly which ones are contributing to worse update practice in IoT devices, compared to general purpose computing devices. Future studies could try to figure this out—for example, by asking IoT Inspector users about their actual update behaviors, thus collecting human-centered data alongside the User Agent data—so that efforts could be directed in the right direction to improve IoT ecosystem update practice.

We recommend IoT device users to install updates promptly upon notification. There are some IoT devices that don't require user interaction after installation, users should mindfully check for updates for these kinds of devices and install them promptly when available. We recommend vendors to use the latest versions of software components, even in cases when a component has few vulnerabilities (for e.g. OkHttp) because with one critical vulnerability attackers could do as much of damage as a software component having multiple critical vulnerabilities. We would also recommend vendors to keep SBOMs and track updates provided by software components they use. Learning from Linux vendors [31, 32, 49] , when software components provide updates, vendors should deploy those updates as soon as possible instead of acting as gatekeeper and blocking them because there is no other way for IoT device users to update those components. Vendors should deploy updates on IoT devices securely using standards like [5, 22], and silently (if possible) as previous studies have pointed out it's the most effective way [17, 20, 21].

6 SUMMARY

We conduct a preliminary study of update practice of IoT devices and find that it's different from the findings of studies conducted about update practices of general purpose computing devices. We find that IoT devices are slower to update to newer versions compared to general purpose computing devices. We couldn't find the exact reason behind this but we narrowed down our findings to where future research should direct their efforts in order to find out which of three factors, end user promptness to install updates, vendor deployment practice, and vendor deployment mechanism are responsible. We also find that when IoT vendors deploy updates they don't update to the latest version, leaving hosts vulnerable to publicly known vulnerabilities.

ACKNOWLEDGMENTS

We would like to thank Santiago Torres Arias and our shepherd Sofía Celi for helpful feedback.

REFERENCES

[1] 2021. The 1,200 IoT companies that are creating the connected world of the future – IoT Startup Landscape 2021. https://iot-analytics.com/iot-startup-landscape/

[2] 2022. Browser detection using the user agent. https://developer.mozilla.org/en-US/docs/Web/HTTP/Browser_detection_using_the_user_agent

[3] 2022. Fingerprinting OpenSSL libraries on IoT devices. https://medium.com/all-things-inspected/fingerprinting-openssl-libraries-on-iot-devices-1d214b4d643

[4] 2022. p0f v3 (3.09b). https://lcamtuf.coredump.cx/p0f3/

[5] 2022. Software Updates for Internet of Things (suit). https://datatracker.ietf.org/wg/suit/about/

[6] O.H. Alhazmi and Y.K. Malaiya. 2005. Modeling the vulnerability discovery process. In *16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05)*. 10 pp.–138. https://doi.org/10.1109/ISSRE.2005.30

[7] O.H. Alhazmi, Y.K. Malaiya, and I. Ray. 2007. Measuring, analyzing and predicting security vulnerabilities in software systems. *Computers & Security* 26, 3 (2007), 219–228. https://doi.org/10.1016/j.cose.2006.10.002

[8] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *USENIX Security Symposium*.

[9] W.A. Arbaugh, W.L. Fithen, and J. McHugh. 2000. Windows of vulnerability: a case study analysis. *Computer* 33, 12 (2000), 52–59. https://doi.org/10.1109/2.889093

[10] Ashish Arora, Ramayya Krishnan, Rahul Telang, and Yubao Yang. 2004. Impact of Vulnerability Disclosure and Patch Availability - An Empirical Analysis. In *In Third Workshop on the Economics of Information Security*.

[11] Meriem Bettayeb, Qassim Nasir, and Manar Abu Talib. 2019. Firmware Update Attacks and Security for IoT Devices: Survey. In *Proceedings of the ArabWIC 6th Annual International Conference Research Track* (Rabat, Morocco) *(ArabWIC 2019)*. Association for Computing Machinery, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.1145/3333165.3333169

[12] Leyla Bilge, Yufei Han, and Matteo Dell'Amico. 2017. RiskTeller: Predicting the Risk of Cyber Incidents. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1299–1311. https://doi.org/10.1145/3133956.3134022

[13] Hasan Cavusoglu, Huseyin Cavusoglu, and Srinivasan Raghunathan. 2005. Emerging Issues in Responsible Vulnerability Disclosure. In *WEIS*.

[14] Sandy Clark, Michael Collis, Matt Blaze, and Jonathan M. Smith. 2014. Moving Targets: Security and Rapid-Release in Firefox. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (Scottsdale, Arizona, USA) *(CCS '14)*. Association for Computing Machinery, New York, NY, USA, 1256–1266. https://doi.org/10.1145/2660267.2660320

[15] "cURL". 2022. *"Command line tool and library for transferring data with URLs"*. "https://curl.se/"

[16] Louis F. DeKoven, Audrey Randall, Ariana Mirian, Gautam Akiwate, Ansel Blume, Lawrence K. Saul, Aaron Schulman, Geoffrey M. Voelker, and Stefan Savage. 2019.

Measuring Security Practices and How They Impact Security. In *Proceedings of the Internet Measurement Conference* (Amsterdam, Netherlands) *(IMC '19)*. Association for Computing Machinery, New York, NY, USA, 36–49. https://doi.org/10.1145/3355369.3355571

[17] Thomas Duebendorfer and Stefan Frei. 2009. Web Browser Security Update Effectiveness. In *Proceedings of the 4th International Conference on Critical Information Infrastructures Security* (Bonn, Germany) *(CRITIS'09)*. Springer-Verlag, Berlin, Heidelberg, 124–137.

[18] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, and J. Alex Halderman. 2014. The Matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference* (Vancouver, BC, Canada) *(IMC '14)*. Association for Computing Machinery, New York, NY, USA, 475–488. https://doi.org/10.1145/2663716.2663755

[19] "Fingerbank". 2022. *"ACCURATELY IDENTIFY CONNECTED DEVICES PERFORM ANOMALY DETECTION"*. "https://www.fingerbank.org/about/"

[20] Stefan Frei, Thomas Duebendorfer, and Bernhard Plattner. 2009. Firefox (In) Security Update Dynamics Exposed. *SIGCOMM Comput. Commun. Rev.* 39, 1 (dec 2009), 16–22. https://doi.org/10.1145/1496091.1496094

[21] Christos Gkantsidis, Thomas Karagiannis, and Milan VojnoviC. 2006. Planet Scale Software Updates. *SIGCOMM Comput. Commun. Rev.* 36, 4 (aug 2006), 423–434. https://doi.org/10.1145/1151659.1159961

[22] Russ Housley. 2005. Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages. RFC 4108. https://doi.org/10.17487/RFC4108

[23] Danny Yuxing Huang, Noah Apthorpe, Frank Li, Gunes Acar, and Nick Feamster. 2020. IoT Inspector: Crowdsourcing Labeled Network Traffic from Smart Home Devices at Scale. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 2, Article 46 (jun 2020), 21 pages. https://doi.org/10.1145/3397333

[24] Amazon.com Inc. 2022. *Amazon Alexa*. https://en.wikipedia.org/wiki/Amazon_Alexa

[25] SmartThings Inc. 2022. *Samsung SmartThings*. https://en.wikipedia.org/wiki/SmartThings

[26] Wolfgang Kandek. 2009. *The Laws of Vulnerabilities 2.0*. https://www.qualys.com/docs/laws-of-vulnerabilities-2.0.pdf

[27] Moazzam Khan, Zehui Bi, and John A. Copeland. 2012. Software updates as a security metric: Passive identification of update trends and effect on machine infection. In *MILCOM 2012 - 2012 IEEE Military Communications Conference*. 1–6. https://doi.org/10.1109/MILCOM.2012.6415869

[28] Dae-Young Kim, Seokhoon Kim, and Jong Hyuk Park. 2018. Remote Software Update in Trusted Connection of Long Range IoT Networking Integrated With Mobile Edge Cloud. *IEEE Access* 6 (2018), 66831–66840. https://doi.org/10.1109/ACCESS.2017.2774239

[29] Deepak Kumar, Kelly Shen, Benton Case, Deepali Garg, Galina Alperovich, Dmitry Kuznetsov, Rajarshi Gupta, and Zakir Durumeric. 2019. All Things Considered: An Analysis of {IoT} Devices on Home Networks. In *28th USENIX security symposium (USENIX Security 19)*. 1169–1185.

[30] Antonio Langiu, Carlo Alberto Boano, Markus Schuß, and Kay Römer. 2019. UpKit: An Open-Source, Portable, and Lightweight Update Framework for Constrained IoT Devices. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. 2101–2112. https://doi.org/10.1109/ICDCS.2019.00207

[31] Ben Laurie. 2008. *Debian and OpenSSL: The Aftermath*. https://www.links.org/?p=328

[32] Ben Laurie. 2008. *Vendors Are Bad For Security*. https://www.links.org/?p=327

[33] Hooman Mohajeri Moghaddam, Gunes Acar, Ben Burgess, Arunesh Mathur, Danny Yuxing Huang, Nick Feamster, Edward W Felten, Prateek Mittal, and Arvind Narayanan. 2019. Watching you watch: The tracking ecosystem of over-the-top tv streaming devices. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 131–147.

[34] Philipp Morgner, Christoph Mai, Nicole Koschate-Fischer, Felix Freiling, and Zinaida Benenson. 2020. Security Update Labels: Establishing Economic Incentives for Security Patching of IoT Consumer Products. In *2020 IEEE Symposium on Security and Privacy (SP)*. 429–446. https://doi.org/10.1109/SP40000.2020.00021

[35] Antonio Nappa, Richard Johnson, Leyla Bilge, Juan Caballero, and Tudor Dumitras. 2015. The Attack of the Clones: A Study of the Impact of Shared Code on Vulnerability Patching. In *2015 IEEE Symposium on Security and Privacy*. 692–708. https://doi.org/10.1109/SP.2015.48

[36] Stephan Neuhaus, Thomas Zimmermann, Christian Holler, and Andreas Zeller. 2007. Predicting Vulnerable Software Components. In *Proceedings of the 14th ACM Conference on Computer and Communications Security* (Alexandria, Virginia, USA) *(CCS '07)*. Association for Computing Machinery, New York, NY, USA, 529–540. https://doi.org/10.1145/1315245.1315311

[37] NVD. 2022. *National Vulnerability Database*. NIST. https://nvd.nist.gov/

[38] "OkHttp". 2022. *"OkHttp HTTP client"*. "https://square.github.io/okhttp/"

[39] Muhammad Talha Paracha, Daniel J Dubois, Narseo Vallina-Rodriguez, and David Choffnes. 2021. IoTLS: understanding TLS usage in consumer IoT devices. In *Proceedings of the 21st ACM Internet Measurement Conference*. 165–178.

[40] Terry Ramos. 2006. *The Laws of Vulnerabilities*. https://www.qualys.com/docs/laws-of-vulnerabilities-presentation.pdf

[41] Robert W. Reeder, Iulia Ion, and Sunny Consolvo. 2017. 152 Simple Steps to Stay Safe Online: Security Advice for Non-Tech-Savvy Users. *IEEE Security and Privacy* (2017).

[42] Jingjing Ren, Daniel J. Dubois, David R. Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. 2019. Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach. In *Proceedings of the Internet Measurement Conference*. 267–279.

[43] "Python Requests". 2022. *"an elegant and simple HTTP library for Python, built for human beings"*. "https://en.wikipedia.org/wiki/Requests_(software)"

[44] Eric Rescorla. 2003. Security Holes . . . Who Cares?. In *12th USENIX Security Symposium (USENIX Security 03)*. USENIX Association, Washington, D.C. https://www.usenix.org/conference/12th-usenix-security-symposium/security-holes-who-cares

[45] Eric Rescorla. 2005. Is Finding Security Holes a Good Idea? *IEEE Security and Privacy* 3, 1 (jan 2005), 14–19. https://doi.org/10.1109/MSP.2005.17

[46] Armin Sarabi, Ziyun Zhu, Chaowei Xiao, Mingyan Liu, and Tudor Dumitras. 2017. Patch Me If You Can: A Study on the Effects of Individual User Behavior on the End-Host Vulnerability State. In *Proceedings of the 18th Passive and Active Measurement PAM*. ACM, Sydney, Australia.

[47] Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X. Liu. 2012. A large scale exploratory analysis of software vulnerability life cycles. In *2012 34th International Conference on Software Engineering (ICSE)*. 771–781. https://doi.org/10.1109/ICSE.2012.6227141

[48] "Wget". 2022. *"Retrieve files via HTTP or FTP"*. "https://en.wikipedia.org/wiki/Wget"

[49] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. 2009. When Private Keys Are Public: Results from the 2008 Debian OpenSSL Vulnerability. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement* (Chicago, Illinois, USA) *(IMC '09)*. Association for Computing Machinery, New York, NY, USA, 15–27. https://doi.org/10.1145/1644893.1644896

[50] Alexander Yohan and Nai-Wei Lo. 2018. An Over-the-Blockchain Firmware Update Framework for IoT Devices. In *2018 IEEE Conference on Dependable and Secure Computing (DSC)*. 1–8. https://doi.org/10.1109/DESEC.2018.8625164

[51] Narges Yousefnezhad, Avleen Malhi, and Kary Främling. 2020. Security in product lifecycle of IoT devices: A survey. *Journal of Network and Computer Applications* 171 (2020), 102779. https://doi.org/10.1016/j.jnca.2020.102779

[52] Chi Zhang, Wonsun Ahn, Youtao Zhang, and Bruce R. Childers. 2016. Live code update for IoT devices in energy harvesting environments. In *2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. 1–6. https://doi.org/10.1109/NVMSA.2016.7547182

A PLOTS

A.1 Avoidable vulnerabilities vs versions behind

Figures in the Appendix A.1 shows the plots for avoidable vulnerabilities vs versions behind for Wget, OkHttp, and Python Requests in Figure 5. The number of avoidable vulnerabilities and CDF of Wget, OkHttp, and Python request shows similar trend to Curl, see Figure 2. The shape of OkHttp and Requests appear to be different because they have fewer vulnerabilities, and that results in zoomed in Y axis; possibly because OkHttp and Requests are written in memory safe languages Java, and Python respectively. Otherwise, OkHttp and Requests also show a long tail distribution. To list the software components in order form most vulnerable to secure, we could look at number of avoidable vulnerabilities in those components, and from that Curl seems most vulnerable, then Wget, Python Requests, and finally OkHttp.

By looking at the plots of Python Requests and OkHttp, one could make an argument that if software component has less vulnerabilities, using newer versions doesn't result in improved the security, but those few vulnerabilities in older version could be exploited by attackers and do as much damage as having large number of vulnerabilities. Unless a software has no vulnerabilities, it's always beneficial to use the latest version.

A.2 Distribution of avoidable vulnerabilities, versions behind, and months behind

Figures in the A.2 shows the distribution of distribution of number of avoidable vulnerabilities, number of versions behind, and

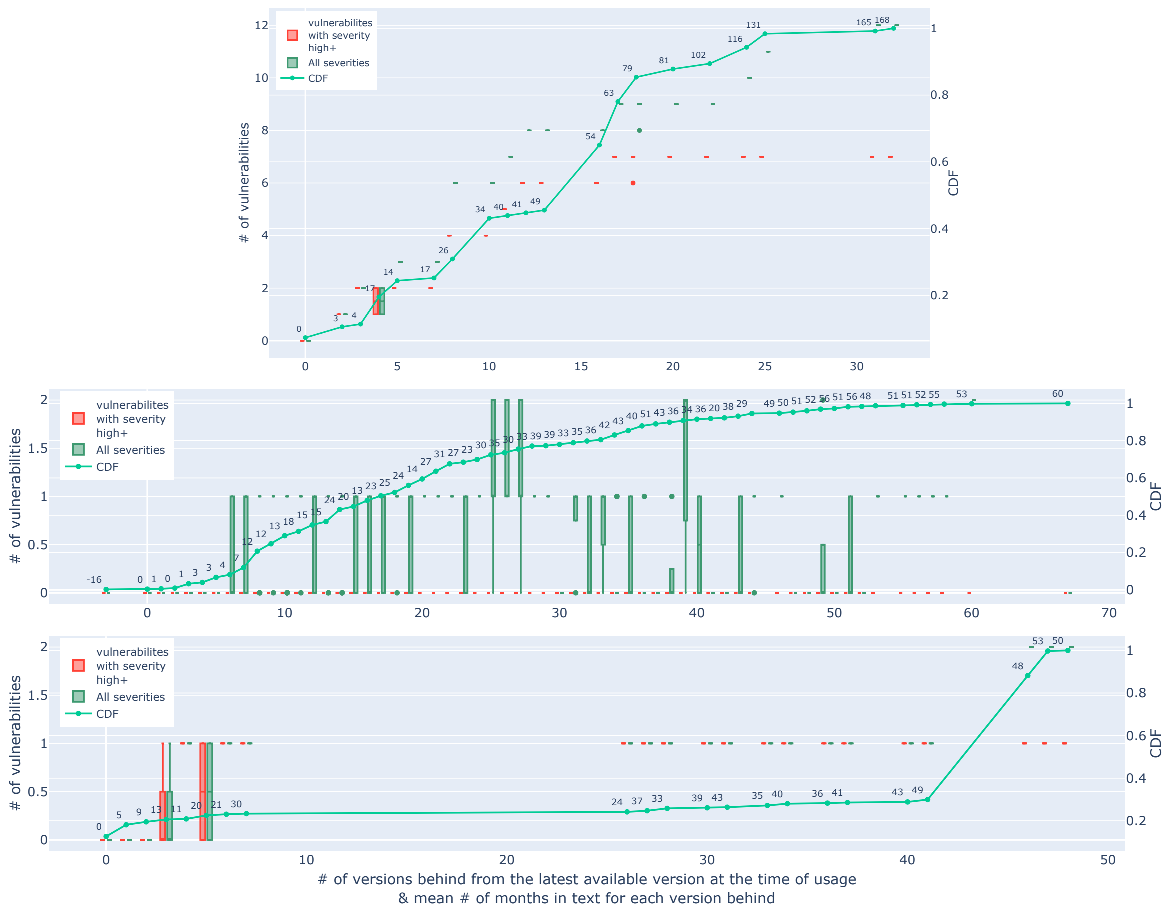

Figure 5: Number of avoidable vulnerabilities in Wget, OkHttp, and Python Requests, from top to bottom, against number of versions behind, and CDF distribution of devices for number of versions behind.

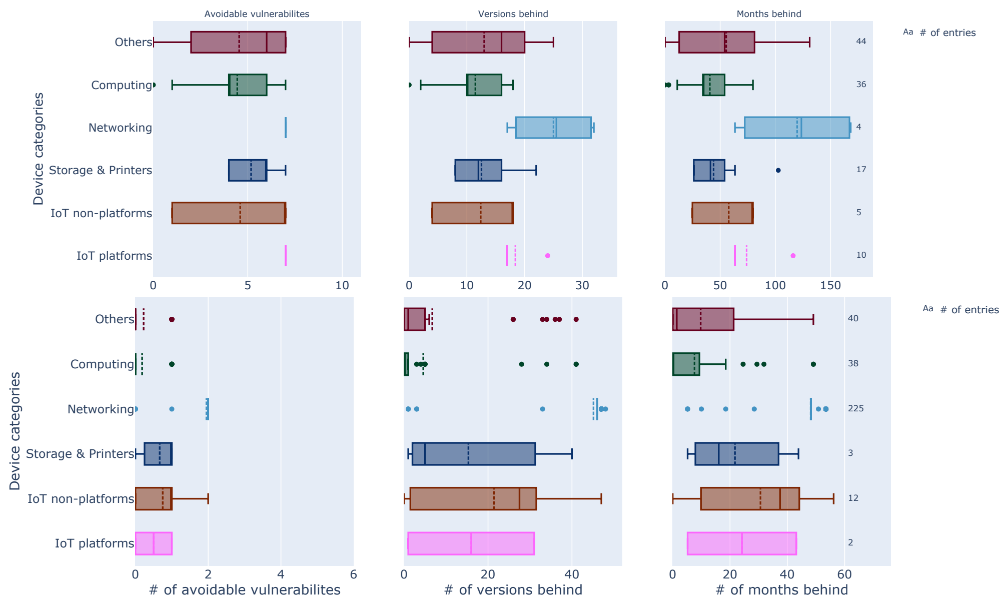

Figure 6: Distribution of number of avoidable vulnerabilities, number of versions behind, and number of months behind for all categories of IoT devices using Wget (top), and Python Requests (bottom).

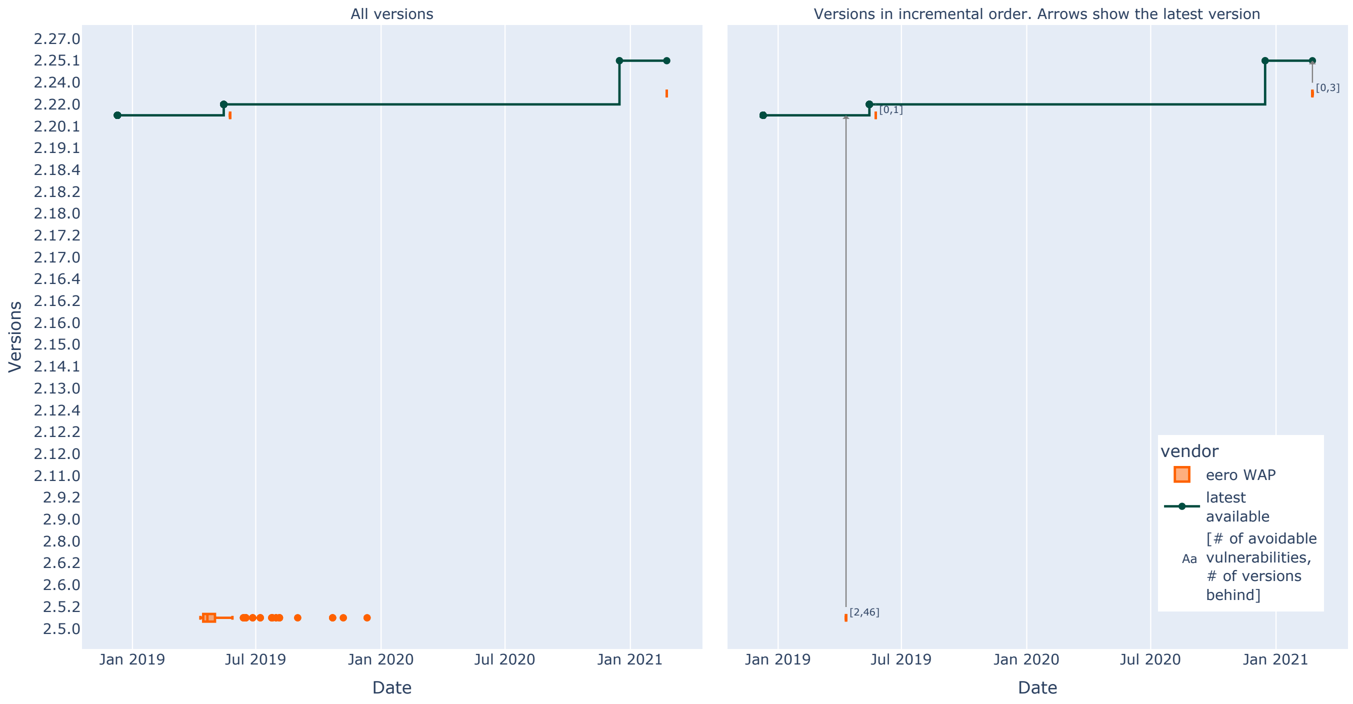

Figure 7: For different vendors using Requests, distribution of versions used against time in left subplot; version seen in incremental order with respect to time, along with avoidable vulnerabilities and version behind in brackets in right subplot.

Table 5: Table of mean, and median (in parenthesis) for each categories of IoT devices' distribution of number of avoidable vulnerabilities, number of versions behind, and number of months behind for all software.

prog	metric	IoT platforms	IoT non-platforms	Storage & Printers	Networking	Computing	Others
cURL	avoidable hi+ vulnerabilities	30.67 (34.00)	10.22 (8.00)	21.83 (16.00)	21.05 (18.00)	15.54 (16.00)	14.31 (8.00)
	versions behind	40.71 (30.00)	10.58 (6.00)	21.67 (15.00)	36.60 (21.00)	17.23 (15.00)	16.60 (6.00)
	months behind	61.35 (40.10)	17.09 (10.50)	31.50 (23.57)	57.34 (31.20)	25.50 (23.57)	25.45 (10.50)
Wget	avoidable hi+ vulnerabilities	7.00 (7.00)	4.60 (7.00)	5.18 (6.00)	7.00 (7.00)	4.44 (4.00)	4.55 (6.00)
	versions behind	18.40 (17.00)	12.40 (18.00)	12.53 (12.00)	25.00 (25.50)	11.47 (10.00)	13.00 (16.00)
	months behind	73.93 (63.40)	57.75 (79.70)	44.03 (41.23)	119.69 (123.55)	40.64 (34.33)	55.47 (54.03)
OkHttp	avoidable vulnerabilities	0.82 (1.00)	0.65 (1.00)	NaN	0.38 (0.00)	0.63 (1.00)	0.75 (1.00)
	versions behind	25.09 (25.00)	27.38 (28.00)	NaN	22.50 (17.50)	19.82 (19.00)	18.32 (16.00)
	months behind	32.87 (38.60)	34.24 (32.67)	NaN	15.11 (9.97)	20.04 (17.25)	23.26 (19.47)
Requests	avoidable vulnerabilities	0.50 (0.50)	0.75 (1.00)	0.67 (1.00)	1.95 (2.00)	0.18 (0.00)	0.22 (0.00)
	versions behind	16.00 (16.00)	21.42 (27.50)	15.33 (5.00)	45.13 (46.00)	4.58 (0.00)	6.72 (1.00)
	months behind	24.17 (24.17)	30.63 (37.47)	21.72 (16.07)	48.15 (48.27)	7.61 (0.00)	9.76 (1.42)

number of month behind for different categories for Wget, and Python Requests in Figure 6. From both the plots we can see that distribution for different categories are different. Looking at the mean and median, update practices of computing group is better than both IoT non-platforms and IoT platforms.

A.3 Versions of software components used by vendors

Figures in the appendix A.3 depicts the version of Python Requests used by vendor eero Wap. Similar to Figure 4 for cURL and OkHttp, we see that vendor has rolled out an updates to versions 2.21.0 and 2.23.0 and these updates are 1 and 3 versions behind from latest available at that time respectively. We have not included the plots for Wget because we did not see any vendor updating Wget in our data. Plot of Python Requests in Figure 7

B STATISTICAL VALUES OF METRICS FOR ALL SOFTWARE COMPONENTS

Table 5 lists the mean and median values of metrics used by us.